REMARKS/ARGUMENTS

Applicant resubmits herewith a copy of the missing PTO/SB/08a and respectfully requests that the Examiner consider the reference identified therein.

In the Office Action, the Examiner has rejected claims 7, 13, 19 and 22 under 35 U.S.C. § 103(a) as being unpatentable over *Peter Haggar* in view of US Patent No. 6,654,778 (*Blandy et al.*). Claims have also been rejected under 35 U.S.C. 112. These rejections are fully traversed below.

Rejection of claims under 35 U.S.C. 112

It is respectfully requested that the Examiner consider reviewing fundamental concepts of computer science. In particular, the Examiner's attention is respectfully directed to strings and string representation which are " such an important and useful datatype that they are implemented in nearly every programming language" (see, for example, the defination of a string in the Wikipedia, the free encyclopedia, a copy of wich is provided in Appendix B for the Examiner's convenience). As previously submitted by the Applicant: "representations of strings depend heavily on the choice of character repertoire and the method of character encoding." (Wikipedia, the free encyclopedia) Wikipedia, the free encyclopedia, provides an example of a null-terminated string stored in a 10-byte buffer, along with its ASCII representation:

F	R	A	N	K	NUL	k	E	F	W
46	52	41	4E	4B	00	6B	65	66	77

Again, it should be noted that Fig. 3 of the present application depicts a method 300 for representing a Java object as a string in accordance with one embodiment of the invention. With reference to Fig. 3, the specification states that at

operation 302, a reference to a Java object is pushed on the execution stack. Next, at operation 304, an inventive Java Bytecode operation is executed. The inventive Java Bytecode operation is designated to represent the object as a string. It should be noted that the Java object is referenced by the reference pushed on the execution stack (operation 302). The specification further states that at operation 306, the string representation of the Java object is determined using the reference to the Java object.

Referring to Fig. 2A, a reference to a Java object (reference A) is depicted in execution stack 212. The specification states that when the inventive Java "to-string" Bytecode instruction 208 is executed, the string representation of the Java object referenced by reference A is determined (Specification, paragraph 24).

It is respectfully submitted that those skilled in the art know that a reference to an object can be used to access that object. It is respectfully requested that the Examiner challenge this assertion for the record if the Examiner believes otherwise. Further, those skilled in the art know that one or more fields of the object can be accessed by using the reference to the object (see, for example, fields 1, 2 and 3 of a Java object 210 shown in Fig. 2A of the present application). It is respectfully requested that the Examiner challenge this assertion for the record if the Examiner believes otherwise. As such, when a field is accessed, binary data of the field (e.g., "100") can be accessed and interpreted in accordance with various conventions to represent it as a string. For example, binary value "100" can represent an integer or a character (e.g., "4", or "A"). As noted in the specification and known in the art, often there is a need to represent a Java object as a string of characters. For example, in order to print an integer object (or an integer field of an object), there is a need to represent the integer as a string of characters (Specification, paragraph 9).

The specification states and those skilled in the art well know that Java programs are generally platform independent (Specification, page 2). As such, an

actual string representation may be dependent on the particular virtual machine implementation and/or the hardware and/or operating system. Thus, one of skilled in the art will know that when an object is accessed, the digital data stored for a field of the object can be represented in accordance with a particular standard (e.g., ANSI, EBCIDIC) that may apply to the particular hardware and/or operating system that runs the Java virtual machine. Accordingly, it is respectfully submitted that one of ordinarily skilled in the art knows that an object and/or a field of an object can be represented as a string of characters (e.g., "10," "Hello World") in accordance with various known standards (e.g., ANSI, EBCDIC). However, conventionally, a Java method is invoked by the virtual machine to represent objects as string characters (Specification, paragraph 9). The claimed invention pertains to a Java Bytecode instruction that effectively determines a string representation of a Java object by using a reference to the Java object stored on an execution stack.

Rejection of the claims under 35 U.S.C. 102

In the Office Action, the Examiner has asserted that the "getfield" Bytecode described by *Haggar* teaches a Bytecode instruction that determines a string representation associated with a Java object (Office Action, page 7). It is noted that *Haggar* states that a "getfield" opcode is used to fetch a field from an object. More particularly, when a "getfield # 5" is executed, "the top value from the stack is popped," and "then the # 5 is used to build an index into the runtime constant pool of the class where the reference to name is stored. When this reference is fetched, it is pushed onto the operand stack" (*Haggar*, page 3).

However, it is respectfully submitted that the "getfield" operation described by *Haggar* merely fetches a field from an object. As such, the "getfield" opcode does NOT additionally determine a string representation for the fetched field. Accordingly, it is respectfully submitted that the Examiner's rejection of claim 1 under 35 U.S.C. 102(a) is improper and should be withdrawn.

Based on the foregoing, it is submitted that the claims are patentably distinct over the cited art of record. Additional limitations recited in the independent claims or the dependent claims are not further discussed because the limitations discussed above are sufficient to distinguish the claimed invention from the cited art. Accordingly, Applicant believes that all pending claims are allowable and respectfully requests a Notice of Allowance for this application from the Examiner.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 500388 (Order No. SUN1P843). Should the Examiner believe that a telephone conference would expedite the prosecution of this application, the undersigned can be reached at the telephone number set out below.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

/RMahboubian/
R. Mahboubian
Registration No. 44,890

September 14, 2006

P.O. Box 70250
Oakland, CA 94612-0250
(650) 961-8300

APPENDIX B

String (computer science)

From Wikipedia, the free encyclopedia

In computer programming and some branches of mathematics, **strings** are sequences of various simple objects. These simple objects are selected from a predetermined set, each entry of which is usually allocated a code. Most commonly these simple objects will be printable characters and the control codes that are used with them. The data types in which these are stored are also called strings and it is fairly common to use these types to store arbitrary, variable-length sequences of binary data. Generally, a string of characters can be placed directly in the source code usually by surrounding it with some form of quote marks—usually ' or ", as these are typeable on most keyboards worldwide. Sometimes the term **binary string** is used to refer to an arbitrary sequence of bits.

Contents

String datatypes

A **string datatype** is a datatype modeled on the idea of a formal string. Strings are such an important and useful datatype that they are implemented in nearly every programming language. In some languages they are available as primitive types and in others as composite types. The syntax of most high-level programming languages allows for a string, usually quoted in some way, to represent an instance of a string datatype; such a meta-string is called a *literal* or *string literal.*

Although formal strings can have an arbitrary (but finite) length, the length of strings in real languages is often constrained to an artificial maximum. In general, there are two types of string datatypes: *fixed length strings* which have a fixed maximum length and which use the same amount of memory whether this maximum is reached or not, and *variable length strings* whose length is not arbitrarily fixed and which use varying amounts of memory depending on their actual size. Most strings in modern programming languages are variable length strings. Despite the name, even variable length strings are limited in length; although, generally, the limit depends only on the amount of memory available.

Historically, string datatypes had one byte for each character, and although the exact character set varied by region, the character encodings were similar enough that programmers could generally get away with ignoring this — groups of character sets used by the same system in different regions either had a character in the same place, or did not have it at all. Mostly these character sets were based on ASCII, though IBM's mainframe systems went their own way and used EBCDIC.

Logographic languages such as Chinese, Japanese and Korean (known collectively as CJK) need far more than 256 characters — the limit of a one-byte-per-character encoding — for reasonable representation. The normal solutions involved keeping single-byte representations for ASCII and using two-byte representations for CJK ideographs. Use of these with existing code led to problems with matching and cutting of strings, the severity of

which depended on how the character encoding was designed. Some encodings such as the EUC family guarantee that a byte value in the ASCII range will only represent that ASCII character making the encoding safe for systems that use those characters as field separators or similar. Others such as ISO-2022 and shift-jis do not make such guarantees, making matching on byte codes unsafe. Another issue is that if the beginning of a string is cut off, important instructions for the decoder or information on position in a multibyte sequence may be lost. Another issue is that if strings are joined together (especially after having their ends truncated by code not aware of the encoding), the first string may not leave the encoder in a state suitable for dealing with the second string.

Unicode has complicated the picture somewhat. Most languages have a datatype for Unicode strings (usually UTF-16 as it was usually added before Unicode supplemental planes were introduced). Converting between Unicode and local encodings requires an understanding of the local encoding, which may be problematic for existing systems where strings of various encodings are being transmitted together with no real marking as to what encoding they are in.

Some languages like C++ implement strings as templates that can be used with any primitive type, but this is the exception, not the rule.

Representations

Representations of strings depend heavily on the choice of character repertoire and the method of character encoding. Older string implementations were designed to work with repertoire and encoding defined by ASCII, or more recent extensions like the ISO 8859 series. Modern implementations often use the extensive repertoire defined by Unicode along with a variety of complex encodings such as UTF-8 and UTF-16.

Most string implementations are very similar to variable-length arrays with the entries storing the character codes of corresponding characters. The principal difference is that, with certain encodings, a single logical character may take up more than one entry in the array. This happens for example with UTF-8, where single characters can take anywhere from one to four bytes. In these cases, the logical length of the string differs from the logical length of the array.

The length of a string can be stored implicitly by using a special terminating character; often this is the null character having value zero, a convention used and perpetuated by the popular C programming language. Hence, this representation is commonly referred to as C string. The length of a string can also be stored explicitly, for example by prefixing the string with byte value — a convention used in Pascal; consequently some people call it a P-string.

In terminated strings, the terminating code is not an allowable character in any string.

Here is an example of a **null-terminated string** stored in a 10-byte buffer, along with its ASCII representation:

F	R	A	N	K	NUL	k	e	f	w
46	52	41	4E	4B	00	6B	65	66	77

The length of a string in the above example is 5 characters, but it occupies 6 bytes. Characters after the terminator do not form part of the representation; they may be either part of another string or just garbage. (Strings of this form are sometimes called *ASCIZ strings*, after the assembly language directive used to declare them.)

Here is the equivalent (old style) **Pascal string** stored in a 10-byte buffer, along with its ASCII representation:

length	F	R	A	N	K	k	f	f	w
05	46	52	41	4E	4B	6B	66	66	77

While these representations are common, others are possible. Using ropes makes certain string operations, such as insertions, deletions, and concatenations more efficient.

Vectors

While character strings are very common uses of strings, a string in computer science may refer generically to any vector of homogenously typed data. A string of bits or bytes, for example, may be used to represent data retrieved from a communications medium. This data may or may not be represented by a string-specific data type, depending on the needs of the application, the desire of the programmer, and the capabilities of the programming language being used.

String algorithms

There are many algorithms for processing strings, each with various tradeoffs. Some categories of algorithms include

- string searching algorithms for finding a given substring or pattern;
- string manipulation algorithms;
- sorting algorithms;
- regular expression algorithms; and
- parsing a string.

Advanced string algorithms often employ complex mechanisms and data structures, among them suffix trees and finite state machines.

Character string oriented languages and utilities

Character strings are such a useful datatype that several languages have been designed in order to make string processing applications easy to write. Examples include the following languages:

- awk
- Icon
- Perl
- Ruby
- Tcl
- MUMPS
- Rexx
- sed
- SNOBOL

Many UNIX utilities perform simple string manipulations and can be used to easily program some powerful string processing algorithms. Files and finite streams may be viewed as strings.

Several string libraries for the C and C++ programming languages do exist which add greater functionality for string processing in those languages:

- The Better String Library (http://bstring.sf.net/)
- The Vstr String Library (http://www.and.org/vstr/)

- Perl Compatible Regular Expressions (http://www.pcre.org/)
- Comparison page for C/C++ string libraries (http://www.and.org/vstr/comparison)

Some APIs like Multimedia Control Interface, embedded SQL or printf use strings to hold commands that will be interpreted.

Recent scripting programming languages, including Perl, Python, Ruby, and Tcl employ regular expressions to facilitate text operations.

Formal theory

One starts with a non-empty finite set Σ called an *alphabet*. Elements of this alphabet are called *characters*. A **string** (or **word**) over Σ is any finite sequence of characters from Σ. Infinite sequences of characters are not allowed in this definition.

A particularly important string is the sequence of no characters, called the **empty string**. The empty string is often denoted ε or λ.

For example, if $\Sigma = \{0, 1\}$, strings over Σ are of the form

ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ...

The set of all strings over Σ is the Kleene closure of Σ and is denoted Σ^*. One can define a binary operation on Σ^* called *concatenation*. If s and t are two strings, their concatenation, denoted st, is defined as the sequence of characters in s followed by the sequence of characters in t.

For example, if s = `bear` and t = `hug` then st = `bearhug` and ts = `hugbear`.

String concatenation is an associative, but non-commutative operation. The empty string serves as the identity element. In algebraic terms, the set Σ^* forms a monoid under string concatenation. In fact, Σ^* is the free monoid generated by Σ.

The *length* of a string is the number of characters in the string. The length can be any natural number. The length of the empty string is 0. Algebraically speaking, the length function defines a monoid homomorphism from Σ^* to **N** (Non-negative integers with addition).

A string s is said to be a **substring** or **factor** of t if there exist two strings u and v such that $t = usv$. One, or both, of u and v can be empty. The relation "is a substring of" defines a partial order on Σ^*, the least element of which is the empty string.

More often, especially in computing applications, one is interested in a different kind of ordering on the set of strings. If the alphabet Σ is well-ordered (cf. alphabetical order) one can define a well ordering on Σ^* called lexicographical order. Note that when Σ is finite, it is always possible to define a well ordering on Σ and thus on Σ^*. For example, the lexicographical ordering of $\{0,1\}^*$ is ε, 0, 1, 00, 01, 10, 11, 000, 001, ...

A set of strings over Σ (i.e. a subset of Σ^*) is called a *formal language* over Σ.

While the alphabet is a finite set and every string has finite length, a language may very well have infinitely many member strings. In fact, Σ^* itself is always an infinite language. Important examples of formal languages include regular expressions and formal grammars.

Character string functions

String functions are used to manipulate a string or change or edit the contents of a string. They also are used to query information about a string. They are usually used within the context of a computer programming language.

The most basic example of a string function is the *length(string)* function. This function returns the length of a string (not counting the null terminator or any other of the string's internal structural information) and does not change the string.

eg. *length("hello world")* would return 11.

There are many string functions which exist in other languages with similar or exactly the same syntax or parameters. For example in many languages the length function is usually represented as *len(string)*. Even though string functions are very useful to a computer programmer, a computer programmer using these functions should be mindful that a string function in one language could behave differently or have a similar or completely different function name, parameters, syntax and outcomes.

Retrieved from "http://en.wikipedia.org/wiki/String_%28computer_science%29"

Categories: Character encoding | Data structures | Data types | Formal languages